Take-Home Assignment: GitHub Actions CI/CD

Objective:

Create and configure GitHub Actions workflows to automate testing and deployment of a simple web project. By the end of this assignment, you should have:

* A workflow to automate testing.
* (Optional) Deployment of a static site to GitHub Pages.

Prerequisites:

1. A GitHub account.
2. A repository with a simple Node.js project or similar, including a test suite.
3. (Optional) A static website (HTML/CSS/JS) to deploy to GitHub Pages.

Part 1: Setup a Test Workflow

Requirements:

1. Trigger the workflow on pushes to the main branch.
2. Actions:
   * Check out the repository.
   * Configure the environment.
   * Install dependencies.
   * Run tests.

* Deliverable: A /.github/workflows/test.yml (or similarly named) file that defines the above workflow.

Part 2: Add a Build/Validation Step

Requirements:

1. Run a build step after the tests pass.
2. Fail the workflow if the build fails.

Deliverable:

* Update the test workflow to include a build step after the test step.

Part 3: Deploy to GitHub Pages (Optional)

Requirements:

1. Deploy only if tests and builds pass and the branch is main.
2. Use GitHub Actions to deploy the static site.

Deliverable:

* Extend the test workflow or create a new workflow /.github/workflows/deploy.yml that deploys the site whenever the main branch is updated and the tests/build pass.

Take-Home Assignment: AWS Architecture Design

Scenario:

Design a simple, production-ready AWS architecture for a new web application consisting of a static front-end, a backend API, and a database.

High-Level Architecture:

1. Front-End Hosting:
   * Amazon S3: Host static website files.
   * Amazon CloudFront: Deliver content globally with low latency.
   * Amazon Route 53: DNS routing to CloudFront distribution.
   * AWS Certificate Manager: SSL/TLS certificates for HTTPS.
2. Backend API:
   * Amazon API Gateway: Expose RESTful API.
   * AWS Lambda: Serverless backend logic.
   * Amazon Cognito: User authentication.
3. Database Layer:
   * Amazon RDS: Managed relational database (MySQL/PostgreSQL).
   * Deployed in Private Subnets with Multi-AZ for high availability.
4. Networking and Security:
   * Amazon VPC: Encompasses the entire architecture.
     + Public Subnets: NAT Gateways.
     + Private Subnets: RDS instances and Lambda functions.
   * IAM Roles: Fine-grained access control.
   * KMS: Encryption for data at rest.
5. CI/CD Pipeline:
   * AWS CodePipeline: Orchestrates build and deployment.
   * AWS CodeBuild: Builds front-end and tests backend.
   * AWS CodeDeploy: Deploys front-end to S3 and updates Lambda functions.
6. Monitoring and Logging:
   * CloudWatch: Logging, metrics, alarms.
   * X-Ray: Request tracing.
   * CloudTrail: Auditing API calls.

Deliverables:

1. Architecture Diagram:
2. CI/CD Pipeline:
   * CodePipeline triggers CodeBuild.
   * Build artifacts are deployed to S3 (front-end) and Lambda (backend).
3. Cost Optimization:
   * Use serverless architecture (Lambda, API Gateway).
   * Optimize S3 with lifecycle policies.
   * Right-size RDS instances.

Take-Home Test: AWS Data Platform Engineer:

You are working as a DevOps engineer for a company that wants to implement a continuous integration pipeline for their serverless application. The application requires several infrastructure and CI/CD setup tasks:

1. **Terraform (Infrastructure Setup):**
   * Provision an S3 bucket named example-bucket.
   * Deploy a Lambda function named example-lambda using Node.js 14.x with an index.handler entry point.
   * Use a local lambda.zip file as the source code, ensuring that its checksum is tracked so any changes trigger redeployment.
   * Set an environment variable LOG\_LEVEL=info for the Lambda function.
2. **Python Script (S3 Upload):**  
   Create a Python script named python\_script.py that uses boto3 to upload a specified file to the newly created S3 bucket. If the upload is successful, log an info-level message, otherwise log an error-level message.
3. **Dockerfile (Containerization):**  
   Write a Dockerfile that:
   * Uses Python 3.9 as the base image.
   * Installs the boto3 library.
   * Copies python\_script.py into the container.
   * Sets the container’s default command to run the Python script.
4. **GitHub Actions (CI Pipeline):**  
   Implement a CI workflow in .github/workflows/ci.yml that:
   * Runs on every push to the main branch.
   * Builds the Docker image you created.
   * Runs the container to test the Python script.

**Question:**  
How would you write the code and configuration files to accomplish all of these tasks? Your solution should include the Terraform configuration, the Python script, the Dockerfile, and the GitHub Actions workflow file.

**SQL Question**

Your database stores transaction records for a retail application in a table called sales\_data. Each record in sales\_data includes a product\_id, an amount sold, and a sale\_date. You’ve been asked by the analytics team to produce a report showing the total amount of sales for each product for the previous calendar month, and then rank the products by their total sales in descending order.

How would you write an SQL query to accomplish this? Consider how you would determine the date range for the previous month’s sales, aggregate sales amounts by product, and then sort the results by total sales.